- メソッド（承前）
- インスタンス変数
- クラス変数とクラスメソッド

★5　クラスの作成（承前）

★5.2　メソッド（承前）

★5.2.2　値を返すメソッド (p.47)

メソッドの定義は，一般に次のような形式である．

アクセス修飾子 | 戻り値の型 | メソッド名 (| 引数 1 の型 | 仮引数 1 |, ..., | 引数 n の型 | 仮引数 n |)
メソッド本体のブロック

アクセス修飾子（上記の例では public）については次節で説明する．前回の HTurtle の例では，house メソッドの戻り値の型は void であり値を返さないが，polygon メソッドの方は下記のように double 型の値を返す．このメソッドの場合，かめさんが引いた線の長さを返すようになっている．右下は，この polygon メソッドの戻り値を使うように T51.java を修正した例である．

☆ 1) return 文はメソッド本体の途中にあってもよいし，条件分岐などのために複数あってもよい．値を返さないメソッドの処理を途中で終わらせるのにも使える．

HTurtle の polygon メソッドの定義

```
public double polygon(int n, double s){
   double a = 360.0/n;
   for(int j = 0; j < n; j++){
     fd(s);
     rt(a);
   }
   return n * s;
}
```

polygon を呼ぶ側の例

```
（T51.java の 11 から 13 行目を修正したもの）
double d = 0;
d += m.polygon(5, size / 2);
m.up(); m.moveTo(100,100,0); m.down();
d += m.polygon(10, size / 5);
System.out.println("d = " + d);
```

return 文の使い方については，C 言語とほぼ同様なので説明を省略する (☆ 1)．

★5.2.3　メソッドの多重定義（オーバーロード） (p.49)

Turtle クラスの moveTo メソッド（p.13 参照）のように，Java では同じ名前で引数の数や型が異なるメソッドを**多重定義（オーバーロード）**することができる．HTurtle.java に以下を追加すると，polygon メソッドを多重定義することになる (☆ 2)．どちらの polygon メソッドが使われるかは，このメソッドを呼ぶ側の引数の指定の仕方（引数の数，型，それらの順序）で決まる．

☆ 2) 追加する場所は，3,4 行目の間，11,12 行目の間，19,20 行目の間のいずれか

辺の長さ s の n 角形をかめの絵の色 c で描く polygon メソッドの定義

```
public double polygon(int n, double s, javafx.scene.paint.Color c){
    javafx.scene.paint.Color prev = tColor;  // 前の色を記憶
    tColor = c;                              // かめの絵の色を c に
    double d = polygon(n, s);                // 2 引数の polygon を呼び出す
    tColor = prev;                           // かめの絵の色を戻す
    return d;                                // 引いた線の長さを返す
}
```

Q1. HTurtle.java に上記の polygon メソッドの定義を追加し，T51.java の 11 行目を次のように変更すると，実行結果はどうなりますか．

```
11    m.polygon(5, size / 2, javafx.scene.paint.Color.RED);
```

★ 5.2.4　メソッドと段階的なプログラムの開発

HTurtle クラスの house メソッドは，右のように定義することもできる．しかし，元のように polygon メソッドを利用して定義した方が分かりやすい，polygon を他にも使いまわせる，機能拡張しやすい，等様々なメリットがある．複雑なプログラムを作る際は，小さな部品を段階的に組み合わせて目的の機能を実現する方がよい．

polygon を使わない house

```
public void house(double s){
    for(int j = 0; j < 4; j++){
        fd(s); rt(90.0);
    }
    fd(s); rt(30);
    for(int j = 0; j < 3; j++){
        fd(s); rt(120.0);
    }
    lt(30); bk(s);
}
```

★ 5.3　インスタンス変数

★ 5.3.1　インスタンス変数の宣言と利用

次は，インスタンス変数を定義して用いることを考える．再び Turtle クラスを拡張して今度は Stepper というクラスを作成する．Stepper クラスの API 仕様は p.51 の通りである．以下の Stepper.java が Stepper クラスを定義するプログラムであり，T52.java がそれを用いるプログラムである．

T52.java

```
 1  import tg.*;
 2
 3  public class T52{
 4      public static void main(String[] args){
 5          TurtleFrame f = new TurtleFrame();

 6          Stepper m1 = new Stepper(); f.add(m1);
 7          Stepper m2 = new Stepper(); f.add(m2);

 8          m1.n = 3;  m1.size = 100;
 9          m2.n = 4;  m2.size = 100;

10          m1.up(); m1.moveTo(100,200,0); m1.down();
11          for(int i = 0; i < 4; i++){
12              m1.step();
13              m2.step();
14          }
15      }
16  }
```

インスタンス変数は名前の通りインスタンス毎に用意される変数であるから，個々の Stepper インスタンス（T52.java の例では m1 と m2）が保持する n，size，j の値はみな別物である．

Stepper.java

```
 1  import tg.*;
 2
 3  public class Stepper extends Turtle{


 4      public int n;           //公開されたインスタンス変数 n の宣言
 5      public double size;     //公開されたインスタンス変数 size の宣言
 6      private int j = 0;      //（非公開）インスタンス変数 j の宣言


 7      public void step() {
 8          if(j >= n)
 9              return;         //描き終えていたならすぐ終了
10          fd(size);
11          rt(360.0/n);
12          j++;                //j の値を 1 増やす
13      }
14  }
```

次に，ローカル変数とインスタンス変数の違いを考えてみよう．右のような例を考えてみればわかるように，ローカル変数は宣言されたブロックの中でしか有効でないため，メソッド内で宣言したローカル変数はメソッド呼び出しのたびに初期化される．これに対して，インスタンス変数はインスタンス生成時に初期化された後ずっと値を保持することになる．この例では，step メソッドが呼ばれるたびに i の値は 0 に初期化されるけれど，j の値は step が呼ばれるたびに増えていく）．

```
public void step() {
   int i = 0;  // ローカル変数
   if(j >= n) return;
   fd(size);  rt(360.0/n);
   System.out.println(i+" "+j);
   i++; j++;
}
```

★ 5.3.2 this

メソッドを実行しているインスタンス自身を指す this というキーワードがある．Stepper.java のインスタンス変数 n やインスタンスメソッド fd は，インスタンスメソッドの定義の中では本来はそれぞれ this.n，this.fd と書くべきところであるが，省略しても差し支えないので略記している (☆ 3)．

☆3) インスタンス変数と仮引数やローカル変数の名前を同じにした場合は，互いに区別できるようにインスタンス変数には必ず this を付けないといけない．

Q2. 以下の this をつけられる所すべてに this をつけなさい．

Stepper.java で this をつけられる所すべてに this をつけてみる

```
 3  public class Stepper extends Turtle{
 4      public int n;
 5      public double size;
 6      private int j = 0;
 7      public void step() {
 8          if(j >= n)
 9              return;
10          fd(size);
11          rt(360.0/n);
12          j++;
13      }
14  }
```

Q3. HTurtle.java 中で this を付けられる所を指摘しなさい．

★ 5.4 クラス変数，クラスメソッド (p.53)

★ 5.4.1 クラス変数，クラスメソッド

以前「クラス変数とクラスメソッド」のところで説明したように，クラス変数とクラスメソッドは，API 仕様の該当項目の先頭に static という修飾子がついている．実際のクラス定義の中でも，static をつけて宣言した変数はクラス変数となり，static をつけて定義したメソッドはクラスメソッドとなる．

Q4. Stepper クラスのインスタンス変数 n は，Stepper.java の中で public int n; と宣言されています．これをクラス変数の宣言にするにはどう書きかえたらよいですか．書き換えると動作はどのように変化しますか．

次の Shop.java は，main とクラスメソッドのみから成るプログラムの例である)．このようなクラスメソッドの使い方は C 言語の関数の使い方と似ている．

Shop.java

```
 1  public class Shop {
 2      static double ritsu = 1.8;          // クラス変数
 3
 4      static int urine(int shiirene){     // クラスメソッド
 5          return (int)(ritsu * shiirene);
 6      }
 7
 8      public static void main(String[] args){
 9          int x = 2000, y;
10          y = Shop.urine(x);   // urine メソッドの呼び出し．"Shop."は省略可
11          System.out.println("仕入値: " + x + "円，売値: " + y + "円");
12          Shop.ritsu = 1.2;   // 変数値の変更．"Shop."は省略可
13          y = Shop.urine(x);
14          System.out.println("仕入値: " + x + "円，売値: " + y + "円");
15      }
16  }
```

このように，インスタンス変数／メソッドの場合は this.[変数名／メソッド名] と書くところを，クラス変数／メソッドの場合は[クラス名].[変数名／メソッド名] と書く．「this.」を省略できる場合があるのと同様に，あるクラスを定義したソースの中でそのクラスのクラス変数やクラスメソッドにアクセスする場合には，「[クラス名].」は省略できる．

次に，上述の Shop クラスを別のプログラムから利用する例を示す．

Maido.java

```
1  public class Maido{
2      public static void main(String[] args){
3          Shop.ritsu = 100.0; // ぼったくり
4          int Tsubo = 100, Tawashi = 5;
5          System.out.println("いらっしゃ〜い．");
6          System.out.println("壷は" + Shop.urine(Tsubo) + "円，束子は"
                              + Shop.urine(Tawashi) + "円でっせ．");
7          System.out.println("え，まけろて？かなんな〜");
8          Shop.ritsu = 80.0; // あこぎ
9          System.out.println("よっしゃ，" + Shop.urine(Tsubo) + "円と"
                              + Shop.urine(Tawashi) + "円でええわ．");
10         System.out.println("はいよ，まいどおおきに．");
11     }
12 }
```

★ 5.4.2 main メソッド

Java のプログラムを実行するには，java コマンドを java Hoge のように実行する (☆ 4)．こうすると，java コマンドは Hoge.class というクラスファイルを探し出してこれを実行する (☆ 5)．java コマンドで起動するプログラムの中には，次のような形で main を必ず書く必要がある．

```
public static void main(String[] args){ ... }
```

これは「main は戻り値なしのクラスメソッドであり公開されている」 ということを意味している．main メソッドは public でなければならない．一方，java コマンドで直接起動されていないクラスファイルの中に main メソッドがあっても，明示的に呼び出されない限りはそれは無視される．

例えば，前節の Shop クラスと Maido クラスの例では，Maido クラスだけでなく，Maido クラスが利用している Shop クラス自身にも main メソッドが定義されている．この場合，java Maido と実行すれば Maido クラスの main メソッドが実行され，java Shop と実行すれば Shop クラスの main メソッドが実行される．

☆ 4) java Hoge 123 abc のように引数をつけて実行することもできる．p.73 参照．

☆ 5) これまで扱ってきたプログラムのようにクラス定義に public をつけている場合，Hoge.class のもとになっているのは Hoge.java というソースファイルである．クラスの定義に public をつけない場合，クラスの名前とそれを記述したソースファイルの名前を一致させる必要がない (一つのソースに複数のクラス定義があってもよい) ので，Hoge.class のソースは Fuga.java という名前かもしれない．その場合，javac Fuga.java すると Hoge.class ができる．